UIUCDCS-R-77-881

UILU-ENG 77 1739

An Algebraic Definition of Knuthian Semantics

by

Laurian M. Chirica
David F. Martin

June 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

An Algebraic Definition of Knuthian Semantics

by

Laurian M. Chirica
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

and

David F. Martin
Computer Science Department
University of California
Los Angeles, California 90024

This research was supported in part by the National Science Foundation, Grant No. MCS 03633 A04 and by the U.S. Energy Research and Development Administration, Contract No. E(04-3)-34, PA 214.

Abstract

This paper presents an algebraic definition of Knuthian semantic systems (K-systems or attribute grammars) with both synthesized and inherited attributes. The approach is based on the initial algebra semantics principle formulated by Goguen, Thatcher, Wagner and Wright. Given a K-system semantic definition for a context-free grammar G, it is shown how to construct a many-sorted algebra $\mathfrak{A}$ such that the semantic mapping from G-derivation trees into their "meaning" is the unique homomorphism from the initial algebra T_G of derivation trees into $\mathfrak{A}$. The practical implications of the algebraic definition of K-systems are discussed, and the combined use of Knuth's original formulation and the algebraic approach for the development of semantic definitions is advocated. The usefulness of the algebraic formulation of K-systems is demonstrated by its application to proving the equivalence of K-systems having the same underlying grammar. It is shown that such proofs may require verifying that a K-system possesses certain properties. To this end, a principle of structural induction on many-sorted algebras is formulated, justified, and applied.

1. Introduction

The purpose of this paper is to present an algebraic treatment of Knuth's approach to the specification of the semantics of context-free languages [14, 15, 16]. Knuthian systems, hereinafter called K-systems, have been called declarative semantics [6, 7, 31, 32], K-grammars [1] and attributed grammars [3, 13, 20, 21]. The latter term seems to dominate in the more recent literature, but it emphasizes a syntactic point of view. We view K-systems as Knuth originally intended, namely as a method for assigning meanings to derivation trees on context-free grammars. This viewpoint is in full agreement with the algebraic approach to semantics.

K-systems have been used in many Computer Science investigations, such as programming language translation [20], program correctness [9], program optimization [24], question-answering systems [25], semantics of programming languages such as SIMULA 67 [31] and PL360 [6], and programming language design [1]. Also, various formal models for programming language syntax and translation related to Knuthian semantics have been proposed [5, 17, 23, 30].

Goguen and Thatcher [11] have given an initial algebra formulation of K-systems in which only synthesized attributes can appear. They leave open the problem of including inherited attributes in an algebraic formulation of K-systems. We propose a solution to this open problem in this paper. Our solution, a formulation of K-systems within the framework of initial algebra semantics, combines the intuitive appeal of K-systems with the theoretical power of algebraic methods. This approach permits a simple and precise definition of K-systems, and has been used by Chirica [4] in proofs of correctness of programming language translators. Although we argue that it is possible to convert any K-system definition (including a circular one) into an equivalent algebraic definition, we examine the more useful alternative of using both

methods for writing semantic definitions. We are thus freed from the combinatorial aspects of K-systems such as tree traversal, topological sorting and especially the circularity problem, while taking full advantage of their intuitive appeal. A more recent version of [11], [12], mentions two unpublished suggestions for solving the problem discussed in this paper. One, by Goguen and Zamfir, is different than our approach; the other, by Burstall, appears to be more related to our work.

In this paper we assume a knowledge of K-systems, many-sorted (heterogeneous) algebras [2, 11], and Scott's lattice-theoretic approach to computation as modified by Gordon [10]. Section 2 briefly reviews the initial algebra approach to semantics. The fundamentals of chain-complete posets and K-systems are discussed in Section 3. Section 4 presents an algebraic formulation of K-systems together with discussion and an example. In Section 5 we discuss the practical implications of the algebraic definition of K-systems, where we also argue in favor of the combined use of Knuth's original formulation and the algebraic approach for the development of semantic definitions. Section 6 demonstrates the usefulness of the algebraic formulation of K-systems by applying it to the problem of proving that two K-systems are equivalent. A principle of structural induction on many-sorted algebras useful in carrying out such equivalence proofs is formulated, justified, and applied.

2. Notation and Mathematical Preliminaries

In this section we briefly review the initial algebra approach to semantics and the elementary theory of complete posets. In the sequel, ε denotes the empty string, Φ the empty set, and N the non-negative integers. N together with its natural order relation is denoted ω.

2.1 Initial Algebra Approach to Semantics

Many-Sorted Algebras

Let $\mathcal{S}$ be a set of symbols called sorts. An $\mathcal{S}$-sorted operator domain (or signature) Σ is a family $\{\Sigma_{w,s}\}_{\langle w,s\rangle \in \mathcal{S}^* \times \mathcal{S}}$ of sets. An element $\sigma \in \Sigma_{w,s}$ is called an operator symbol of type $\langle w,s\rangle$, arity w, (target)sort s, and rank $lg(w)$.†

2.1.1 Definition (Many Sorted Algebra)

Let Σ be an $\mathcal{S}$-sorted operator domain. An ($\mathcal{S}$-sorted) Σ-algebra consists of a family $X = \{X_s\}_{s \in \mathcal{S}}$ of sets together with a function $\sigma_X : X^w \to X_s$ for each $\sigma \in \Sigma_{w,s}$, $w \in \mathcal{S}^*$, $s \in \mathcal{S}$. If $w = s_1 s_2 \cdots s_n$, where $n>0$ and $s_i \in \mathcal{S}$, $1 \le i \le n$, then $X^w \triangleq X_{s_1} \times X_{s_2} \times \cdots \times X_{s_n}$; when $w = \varepsilon$, $X^w = \{\varepsilon\}$. ☒

The sets X_s, $s \in \mathcal{S}$, are called the carriers of the Σ-algebra X (we follow the standard practice of referring to a Σ-algebra and its family of carriers by the same name). The functions σ_X are called operations in the algebra X and are said to have the same type, arity, sort, and rank as the operator symbol σ with which σ_X is associated. An operation σ_X of type $\langle \varepsilon, s\rangle$, is a function $\sigma_X : \{\varepsilon\} \to X_s$, called a constant (or nullary) operation. It is often convenient to identify σ_X with the element $\sigma_X(\varepsilon)$ of X_s; in this case σ_X is called a constant of sort s. Many-sorted algebras (Σ-algebras) are also called heterogeneous algebras [2].

2.1.2 Definition (Homomorphism)

Let X and Y be Σ-algebras. A Σ-homomorphism is a family

† When w is a string, lg(w) denotes the length of w.

$h = \{h_s: X_s \to Y_s\}_{s \in \mathcal{S}}$ of functions (abbreviated $h: X \to Y$) such that for all $\langle w,s \rangle \in \mathcal{S}^* \times \mathcal{S}$, $\sigma \in \Sigma_{w,s}$, and $x \in X^w$,

$$h_s(\sigma_X(x)) = \sigma_Y(h^w(x)) ,$$

where if $w = s_1 s_2 \cdots s_n$, $n>0$, $s_i \in \mathcal{S}$, $1 \le i \le n$, $h^w \triangleq \langle h_{s_1}, \ldots, h_{s_n} \rangle$ and $h^w(x) \triangleq \langle h_{s_1}(x_1), \ldots, h_{s_n}(x_n) \rangle$. When $w = \varepsilon$, $h^w(x) \triangleq \varepsilon$ and the above equation reduces to $h_s(\sigma_X) = \sigma_Y$. ☒

Initial Many-Sorted Algebras

The key property for the application of Σ-algebras to programming language semantics is given by the following.

2.1.3 Definition (Initial Many-Sorted Algebra)

A Σ-algebra X is initial in the category of Σ algebras and Σ-homomorphisms if and only if there exists a unique Σ-homomorphism from X to any other Σ-algebra. ☒

2.1.4 Proposition

For each operator domain Σ, there exists an initial Σ-algebra T_Σ. ☒

The proof of the existence of T_Σ is constructive, and is typical of word algebra constructions [2].

In order to apply the property of initiality to the semantics of context-free languages, we define a class of many-sorted algebras associated with each context-free grammar.

Let $G = \langle V_N, V_T, P, S \rangle$ be a context-free grammar (CFG), and let $V = V_N \cup V_T$. Following [12], we define a class of many-sorted algebras, called G-algebras, associated with G. Define a string mapping $nt: V^* \to V_N^*$ such that nt(x) is the string of nonterminals, in their same order, appearing in x. Let $\Pi = \{\pi_p \mid p \in P\}$ be a set of symbols in one-to-one correspondence with P; for convenience, the symbols of Π will be indexed by production numbers.

We define a V_N-sorted operator domain, also called G to identify it with the grammar G. For all $A \in V_N$, $w \in V_N^*$, let

$$G_{w,A} = \{\pi_p \in \Pi \mid p \in P,\ p = A \to x,\ nt(x) = w\}$$

and $G = \{G_{w,A}\}$. The elements of the initial G-algebra T_G represent the derivation trees on CFG G.

The fundamental precept of initial algebra semantics of context-free languages is that any G-algebra X can provide a semantic definition for the context-free language L(G) with (abstract) syntax T_G; the unique homomorphism $h: T_G \to X$ maps each G-derivation tree into its "meaning" in a syntax-directed manner.

2.1.5 Example

Consider the CFG $G = \langle V_N, V_T, P, S\rangle$, where $V_N = \{S, L, A, B\}$, $V_T = \{a, b\}$, and P consists of

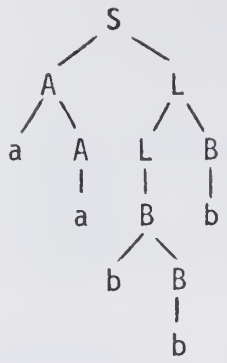
1. $S \to AL$
2. $L \to LB$
3. $L \to B$
4. $A \to aA$
5. $A \to a$
6. $B \to bB$
7. $B \to b$

The initial algebra T_G has carriers $T_{G,D}$ for all $D \in V_N$, where $T_{G,D}$ is the set of linear representations of all D-rooted derivation trees with terminal frontiers. The operations of T_G are defined as follows. Let $A \to x$ be the p-th production, where $nt(x) = A_1A_2\cdots A_n$, $n \geq 0$. The corresponding operation is

$$(\pi_p)_{T_G}(t_1, \ldots, t_n) = \pi_p(t_1 \cdots t_n),\ n>0$$

$$(\pi_p)_{T_G} = \pi_p,\ n = 0,$$

where $t_i \in T_{G,A_i}$, $1 \leq i \leq n$. Thus $\pi_1(\pi_4(\pi_5)\pi_2(\pi_3(\pi_6(\pi_7))\pi_7)) \in T_{G,S}$ is the linear representation of the derivation tree



2.2 Complete Posets

In this section we briefly review some elementary facts from the theory of partially ordered sets (posets). Complete posets and continuous functions thereon are at the foundation of the mathematical apparatus required to resolve certain recursive definitions that arise in Section 4. Before doing so, we remark that Scott [26-29] proposed that continuous lattices should be used in the theory of computation. He offered a unified and general framework for studying recursive definitions, generalizing many previously known results. Following Scott's approach, other authors [10, 11, 19, 22] have shown that Scott's ideas can be applied to more general structures such as posets, with fewer technical difficulties and more intuitive appeal.

Posets

A poset is a nonempty set P together with a reflexive, antisymmetric, and transitive relation $\leq$ on P. A poset will be denoted by its set P, the relation $\leq$ being implicit. An upper bound of a subset X of P is any element $u \in P$ such that $x \leq u$ for all $x \in X$. If $u \leq u'$ for any other upper bound u' of X, then u is said to be the least upper bound (lub) of X in P. We write $\bigsqcup_P X$, $\bigsqcup_{x \in X} x$ or simply $\bigsqcup X$ for the lub of X in P if it exists. If $X = \{x_1, \ldots, x_n\}$ then we write $\bigsqcup_{i=1}^{n} x_i$ to designate $\bigsqcup X$. For a denumerable sequence $\langle x_i \rangle_{i \in \omega}$ of elements in P we write $\bigsqcup_{i=0}^{\infty} x_i$ to designate the lub of the set $\bigcup_{i \in N} \{x_i\}$. If the empty set Φ has a lub in P it is called the least or the

bottom element of P and is denoted $\perp_P$ or simply $\perp$ when there is no danger of confusion. $\perp_P$ has the property that $\perp_P \leq p$ for all $p \in P$. A poset with a least element is called strict.

A subset $X \subseteq P$ of a poset P is a chain in P iff for $x, y \in X$, either $x \leq y$ or $y \leq x$ (total order). The number of elements in a finite chain is called the length of the chain. The height ht(P) of a poset P without infinite chains is the least upper bound (in ω) of the lengths of the chains in P. A poset of finite height is called elementary. The notion of height of an elementary poset has more intuitive appeal if we think of it in relation to the usual graph representation for a poset (Hasse diagram), in which case the height is the number of nodes in the longest upward-oriented path in the graph.

A poset is (chain) complete iff every non-empty chain has a lub. Since any finite chain $x_1 \leq \ldots \leq x_n$, $n > 0$, has a lub $\bigsqcup_{i=1}^{n} x_i = x_n$, it follows immediately that any elementary poset is complete. A chain complete poset will be called a domain. If a complete poset is also strict (elementary) then it is called a strict (elementary) domain. We use the following facts in the sequel:

(a). Any set together with its equality relation is an elementary domain of height 1;

(b) For any set A, the disjoint union $A_\perp = A \cup \{\perp\}$ with the order relation $a \leq b$ iff $a = \perp$ or $a = b$, is a strict elementary domain of height 2;

(c) If $A_1, \ldots, A_n$ are elementary domains, then so is $A_1 \times \ldots \times A_n$, and

$$ht(A_1 \times \ldots \times A_n) = \sum_{i=1}^{n} ht(A_i) - n + 1.$$

Continuous Functions on Posets

Let P and Q be posets. A function $f: P \to Q$ is monotonic iff for all $p, p' \in P$, $p \leq p'$ implies $f(p) \leq f(p')$. A function $f: P \to Q$ is (chain) continuous iff for any non empty chain X with lub in P, the set f(X) has a lub in Q and $f(\bigsqcup_P X) = \bigsqcup_Q f(X)$. The set of all continuous functions from P to Q, denoted $[P \to Q]$, is a complete poset. For every strict and complete poset P there exists a continuous function $Fix_P: [P \to P] \to P$, called the least fixed point operator, defined as

$$Fix_P(f) = \bigsqcup_{i=0}^{\infty} f^i(\perp_P)$$

where $f^0 = 1_P$ (identity function on P) and $f^{i+1} = f \circ f^i$. For strict posets P and Q, a function $f: P \to Q$ is called strict if $f(\perp_P) = \perp_Q$. In the remainder of this paper, completeness means chain completeness and continuity means chain continuity.

3. Knuthian Semantic Systems (K-Systems)

In this section we review and discuss K-systems as originally presented by Knuth. We then impose a few restrictions (which result in no loss of generality) on K-systems which make them more easily adapted to their algebraic formulation in Section 4. The use of the word "domain" in this section generally refers to unordered sets of values, and not to the domains of Section 2.2.

The basic purpose of K-systems is the syntax-directed specification of a mapping from derivation trees on a context-free grammar to corresponding semantic objects (which can be vector-valued). K-systems ignore how derivation trees are obtained from the terminal "source" strings which constitute their frontier. Nevertheless, K-systems are sometimes regarded as defining a relation (since the CFG may be ambiguous) between source strings and semantic objects, as depicted in Figure 1.

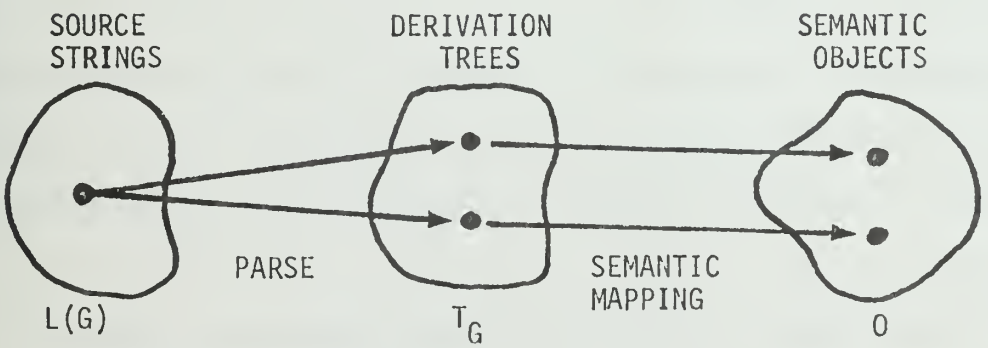


Figure 1 - Relations Defined by K-systems

Informally, K-systems as presented by Knuth in [14] consist of (1): a CFG $G = \langle V_N, V_T, P, S \rangle$ in which S appears in the left part of exactly one production and in the right part of no productions; (2): two disjoint sets of symbols associated with each symbol of G, called inherited and synthesized attributes, where S must have at least one synthesized and no inherited

attributes; (3): a semantic domain D_α for each attribute α; (4): a set of semantic rules associated with each production. Each semantic rule is a function $f_{pk\alpha}: D_{\alpha_1} \times \ldots \times D_{\alpha_t} \rightarrow D_\alpha$ which defines the value of an attribute α of an instance k of a symbol appearing in production p in terms of the values of attributes $\alpha_1, \ldots, \alpha_t$ associated with other instances of symbols in the same production.

In [16], no attributes are associated with terminal symbols. This is a reasonable restriction, since the association of attributes with terminal symbols is unnecessary and even bad because it binds the semantic definition to concrete rather than abstract syntax.

We now discuss how the semantic rules are specified. Let

$$A \rightarrow a_0 B_1 a_1 \ldots B_r a_r, \quad r \geq 0,$$

where $a_0, \ldots, a_r \in V_T^*$ and $A, B_1, \ldots, B_r \in V_N$, be a production (say the p-th) of G (strictly speaking, r depends upon p, but for typographical convenience this dependency is not explicitly indicated). The B_k, $1 \leq k \leq r$, represent distinct nonterminal instances; the same nonterminal may be represented more than once. According to [14, 16], the semantic rules $f_{pk\alpha}$ associated with this production must define

(i) the values of all and only the synthesized attributes of A,

and

(ii) the values of all and only the inherited attributes of B_k, $1 \leq k \leq r$.

To the above requirements we add

(iii) the values of the synthesized attributes of A can be defined only in terms of the values of its own inherited attributes and synthesized attributes of the B_k, $1 \leq k \leq r$;

(iv) the values of the inherited attributes of any B_k, $1 \leq k \leq r$, can be

defined only in terms of inherited attributes of A and synthesized attributes of any B_j, $1 \leq j \leq r$.

We claim the following about conditions (iii) and (iv):

(a) They do not affect the definitional power of K-systems or their convenience in practical use. A constructive and informal argument presented in the Appendix shows that any K-system which does not satisfy these restrictions can be replaced by an equivalent one that does;

(b) Semantic rules which do not satisfy these conditions are unreasonable and infringe upon the very meaning and purpose of introducing "inherited" and "synthesized" attributes.

The language proposed in [14] for defining semantic rules consists of

(a) variable symbols, denoted $\alpha.A$ (Knuth writes $\alpha(A)$), where α is an attribute symbol and A is a nonterminal instance within a given production; $\alpha.A$ denotes attribute α of nonterminal A. Multiple instances of the same nonterminal within a production are distinguished by supplementary marks such as integer subscripts. For example, A_1 and A_2 would both represent the same nonterminal A, and $\alpha.A_1$ and $\alpha.A_2$ both range over D_α;

(b) function (and constant) symbols of various types and arities, interpreted as functions over the semantic domains D_α. We refer to these functions as primitive semantic functions to distinguish them from the semantic functions $f_{pk\alpha}$;

(c) terms, constructed in the usual way from variables and function symbols.

Using this language, the semantic rules are specified as sets of equations of the form "v = t" where v is a variable symbol and t is a term.

Let x.A and y.A be vectors of variable symbols which denote, respectively, the inherited and synthesized attributes of A. This notation is not intended to imply that each nonterminal symbol has the same set of inherited attribute symbols or the same set of synthesized attribute symbols. In accordance with conditions (i) - (iv) above, the semantic rules associated with the above (p-th) production are

$$(1) \qquad \begin{aligned} x.B_1 &= F_{p1}(x.A,\ y.B_1,\ldots,\ y.B_r) \\ &\vdots \\ x.B_r &= F_{pr}(x.A,\ y.B_1,\ldots,\ y.B_r) \end{aligned}$$

$$(2) \qquad y.A = G_p(x.A,\ y.B_1,\ldots,\ y.B_r)$$

When $r = 0$, $A \to a_0$ is called a terminal production; in this case equations (1) are absent and equation (2) reduces to $y.A = G_p(x.A)$. When $A = S$, $x.S$ does not appear in the above equations. A K-system in which only synthesized attributes appear is called purely synthesized.

$F_{pk}(x.A,\ y.B_1,\ldots,\ y.B_r)$ is a vector of terms interpreted to define semantic functions $f_{pk\alpha}$ for all inherited attributes α of B_k, $1 \le k \le r$. The vector of terms $G_p(x.A,\ x.B_1,\ldots,\ x.B_r)$ is similarly interpreted to define semantic functions $f_{p0\alpha}$ for all synthesized attributes α of A. In algebraic terminology, the sets D_α and the primitive semantic functions will form a Σ-algebra for some specified operator domain Σ. The semantic functions are therefore derived operators (cf. [12]) in the algebra. This raises, however, a serious problem which will be discussed in Section 3.4. We now present

an example.

3.1 Example

Consider the following K-system.

Nonterminal	Inherited Attributes	Synthesized Attributes	Semantic Domains
S	-	x	$D_x = \{a,b\}^*$
L	y	x	$D_y = \{a,b\}^*$
A	-	x	
B	-	x	

Productions	Semantic Rules
1. $S \rightarrow AL$	$x.S = x.L \frown x.A$
	$y.L = x.A$
2. $L_1 \rightarrow L_2B$	$x.L_1 = x.L_2 \frown y.L_1 \frown x.B$
	$y.L_2 = y.L_1$
3. $L \rightarrow B$	$x.L = y.L \frown x.B$
4. $A_1 \rightarrow aA_2$	$x.A_1 = a \frown x.A_2$
5. $A \rightarrow a$	$x.A = a$
6. $B_1 \rightarrow bB_2$	$x.B_1 = b \frown x.B_2$
7. $B \rightarrow b$	$x.B = b$

The primitive semantic functions in this K-system are string concatenation ($\frown$) and the string constants "a" and "b". The semantic objects are elements of $D_x = \{a,b\}^*$. This K-system defines the source string-to-semantic object relation

$$\{<a^m b^n, a^m b^{i_1} a^m \cdots a^m b^{i_k} a^m> \mid k,m,n>0,\ i_j>0,\ 1\le j\le k,\ \sum_{j=1}^{k} i_j = n\}.$$

This relation can be regarded as representing all partitions of each positive integer.

3.2 Definition (Equivalence of K-Systems)

Two K-systems are equivalent if and only if they define the same source string-to-semantic object relation. ☒

3.3 Example

Consider the following purely synthesized K-system, equivalent to that of Example 3.1.

Nonterminal	Inherited Attributes	Synthesized Attributes	Semantic Domains
S	-	x	D_x = {a, b}*
L	-	x, z	D_z = {a, b}*
A	-	x	
B	-	x	

Productions	Semantic Rules
1. $S \to L$	$x.S = x.L \frown z.L$
2. $L_1 \to L_2 B$	$x.L_1 = x.L_2 \frown z.L_2 \frown x.B$
	$z.L_1 = z.L_2$
3. $L \to AB$	$x.L = x.A \frown x.B$
	$z.L = x.A$
4. $A_1 \to aA_2$	$x.A_1 = a \frown x.A_2$
5. $A \to a$	$x.A = a$
6. $B_1 \to bB_2$	$x.B_1 = b \frown x.B_2$
7. $B \to b$	$x.B = b$

Note that the grammar underlying this K-system is not the same as that underlying the K-system of Example 3.1.

3.4 Discussion

The problem with defining functions by terms in a certain language is that a term does not quite define a function in the sense that we do not really know the intended domain of definition of that function. For instance, the term "z+1" over integers may define the function $f: Z \to Z$ or the function $f': Z \times Z \to Z$, where $f = \lambda z.z+1$ and $f' = \lambda z_1 z_2.z_1 + 1$. This may not seem very important, but the concept of attribute dependency is important in K-systems and it is based on the specification of the domain of definition of semantic functions. According to Knuth [14, p. 133],

> "(an) attribute α is defined to have the value v ... if, in the corresponding semantic rule
>
> $$f_{pj\alpha}: D_{\alpha_1} \times \cdots \times D_{\alpha_t} \to D_{\alpha}$$
>
> all of the attributes $\alpha_1, \ldots, \alpha_t$ have previously been defined to have the respective values $v_1, \ldots, v_t$ at the respective nodes,... and $v = f_{pj\alpha}(v_1, \ldots, v_t)$."

This means that if $f: Z \times Z \to Z$ is defined as $f(z_1,z_2) = z_1 + 1$, then $f(u,v)$ cannot be computed if u is defined but v is not.

In current practical use of K-systems, which use the specification language described previously, the information about the intended domain of definition of semantic functions is not usually provided. Without the knowledge about the intended domain of definition, we cannot establish the graph model of attribute dependency in the sense proposed by Knuth. Most often this information is deduced on syntactic grounds; e.g., "z+1" defines a function $f: Z \to Z$ of one variable since there is only one variable in the term "z+1". This approach fails if we wish to be able to evaluate the term

"if x>0 then x+1 else y" when $x > 0$ and y is undefined.

The solution to these problems is based on the idea of formalizing the concept of a variable being defined or undefined: we simply add a new value $\perp$ to the sets D_α such that an "undefined" attribute has the value $\perp$. By extending the primitive semantic functions to $D_\alpha \cup \{\perp\}$ we can say that an attribute α_1 directly depends on attribute α_2 iff the semantic function defining α_1 is strict in the argument corresponding to α_2. This does not conform to Knuth's attribute dependency definition, but seems to be better suited for practical applications where we do not want to specify intricate domains of definition. In our definition all semantic functions attached to the same production will have identical domains of definition. These domains need not be specified since they will be implied by the form of the production. By proper extension of the primitive semantic functions any desired dependency among attributes can be achieved (e.g., the primitive semantic function $\lambda pxy.$if p then x else y can be extended in various ways, making it strict in all or some of its arguments).

The last problem to be considered is how to compute the attribute values. This problem is not given a full formal treatment in [14, 16]. In [14] it is indicated that well known tree manipulation algorithms may be used for attribute evaluation by supplementing these algorithms with the necessary attribute computation steps. The work of Fang [7], Bochmann [3], and Kennedy and Warren [18] shows that this is not a trivial task. The main contribution of the algebraic definition given in Section 4 is a precise mathematical definition of this aspect of K-systems. Furthermore, the definition is independent of any tree manipulation algorithm.

4. Algebraic Definition of Knuthian Semantics

In this section we propose an algebraic definition of K-systems [14, 15, 16] and discuss the implications and the use of this definition for semantics and translation of programming languages.

4.1 Definition (K-System)

A Knuthian Semantic System (K-system) consists of:

(a) A context-free grammar $G = \langle V_N, V_T, P, S\rangle$ in which S occurs in the left part of only one production and in the right part of no productions;

(b) A Σ-algebra D for a given set of sorts $\mathcal{S}$ and a given $\mathcal{S}$-sorted operator domain Σ. The carriers D_s, $s \in \mathcal{S}$, of D are called semantic domains and the operations $f \in \Sigma_{w,s}$, $\langle w,s\rangle \in \mathcal{S}^* \times \mathcal{S}$, of D are called primitive semantic functions;

(c) Two functions $\overline{\ }: V_N \to \mathcal{S}^*$ and $\underline{\ }: V_N \to \mathcal{S}^*$, called attribute type assignment rules, which associate with each nonterminal $A \in V_N$ two strings $\overline{A}$ and $\underline{A}$ in $\mathcal{S}^*$ called, respectively, the inherited attribute type and the synthesized attribute type of A. It is assumed that $\overline{S} = \varepsilon$ and $\underline{S} \neq \varepsilon$;

(d) Semantic functions. For every production (say the p-th)

$$A \to a_0 B_1 a_1 \ldots B_r a_r \ \dagger$$

in P, where $a_0, a_1, \ldots, a_r \in V_T^*$, $A, B_1, \ldots, B_r \in V_N$, $r \geq 0$, there is given a function G_p of type $\langle \overline{A}\underline{B_1}\ldots\underline{B_r}, \underline{A}\rangle$ iff $\underline{A} \neq \varepsilon$, and for every k, $1 \leq k \leq r$, there is given a function F_{pk} of type $\langle \overline{A}\underline{B_1}\ldots\underline{B_r}, \overline{B_k}\rangle$ iff $\overline{B_k} \neq \varepsilon$. All semantic functions must be derived operators of the indicated type in algebra D in the sense of [12]. ☒

If $\overline{A} = s_1 \ldots s_n \in \mathcal{S}^+$, then $D^{\overline{A}} = D_{s_1} \times \ldots \times D_{s_n}$. If $\overline{A} = \varepsilon$ then $D^{\overline{A}} = \{\varepsilon\}$. The set $D^{\overline{A}}$ is called the inherited attribute domain of A. $D^{\underline{A}}$, called the

† For typographical convenience, the dependence of r on p is not shown.

synthesized attribute domain of A, is defined similarly. The functions F_{pk} are of type $\langle \overline{A}\underline{B}_1 \ldots \underline{B}_r, \overline{B}_k \rangle$, i.e.,

$$F_{pk}: D^{\overline{A}} \times D^{\underline{B}_1} \times \ldots \times D^{\underline{B}_r} \to D^{\overline{B}_k}.$$

If $\overline{B}_k = s_1 \ldots s_m$, then F_{pk} is a $\overline{B}_k$-tuple of functions $\langle F_{pk1}, \ldots, F_{pkm} \rangle$ where

$$F_{pki}: D^{\overline{A}} \times D^{\underline{B}_1} \times \ldots \times D^{\underline{B}_r} \to D_{s_i}, \quad 1 \le i \le m.$$

The functions G_p are defined similarly. Note that for terminal productions $A \to a_0$ there are no semantic functions F_{pk} and the semantic functions G_p are of type $\langle \overline{A}, \underline{A} \rangle$.

4.2 Specification of K-systems

It is clear that a specification language is built into Definition 4.1. The basic idea is that of assigning variable symbols to every nonterminal instance in a production, according to the nonterminal type as defined by the functions "$\overline{\ }$" and "$\underline{\ }$". Then we use terms of the appropriate type to specify semantic functions. The specification language outlined here is the same as the one discussed in Section 3, only here it results naturally from the algebraic definition. (In a sense, Definition 4.1 formalizes the semantics of Knuth's specification language [14].)

4.3 K-system Semantics Formulated as Initial Algebra Semantics

We must now make precise what attribute computation means, i.e., how to use a K-system for semantic definition of context-free languages. The idea is simple and is based upon the following observation about how a Knuthian definition (in the informal sense) works. For any given nonterminal $A \in V_N$, $A \neq S$, and any given G-derivation tree t^A with root A, the values of the synthesized attributes y.A of node A are obtained in a unique way from the values of its inherited attributes x.A. If the definition is

circular then the value of some attributes cannot be computed (i.e., they are undefined), but those which are defined are uniquely obtained from x.A. This observation is true first because in any Knuthian definition (in the informal sense) if the attribute values can be computed then they are unique [14, p. 135], and second because of conditions (iii) and (iv) laid down in Section 3. The correspondence $x.A \mapsto y.A$ is a partial function. If we use a special value $\perp$ to represent undefined attributes, then the correspondence $x.A \mapsto y.A$ is a total function. Accordingly, we assume that an element $\perp_{D_s}$ has been added to every carrier D_s of Σ-algebra D. We also assume that every D_s, $s \in \mathcal{S}$, is made into a strict elementary domain $(D_s)_\perp$ as indicated in Section 2.2. Finally we assume that the operations of D have been extended to continuous functions. In the terminology of [12], D is a (chain) continuous Σ-algebra. From Proposition 4.13 of [12], derived operators of continuous algebras are continuous. Therefore, the semantic functions 4.1(d) are continuous functions on their respective domains.

In view of the previous discussion, the semantics of a tree t^A with root A in grammar G is a <u>function</u> $\sigma.A \in [D^{\overline{A}} \to D^{\underline{A}}]$ which maps inherited attributes into synthesized attributes of node A. Therefore, the problem is to define the correspondence $t^A \mapsto \sigma.A$ between derivation trees t and functions σ. This discussion applies to trees with roots other than S. For trees with root S the semantics continue to be $y.S \in D^{\underline{S}}$, i.e., the value of the synthesized attributes of node S. We write $\sigma.S$ for uniformity.

An appropriate tool for mapping derivation trees into their semantics is provided by the initial algebra semantics. We define a G-algebra $\mathcal{K}$ with carriers K_A, $A \in V_N$ and operations θ_p, $p \in P$ such that the unique homomorphism $h: T_G \to \mathcal{K}$ maps derivation trees t into semantic objects σ as discussed above.

The choice of carriers has already been made by the previous discussions, i.e., $K_A = [D^{\overline{A}} \to D^{\underline{A}}]$ for all $A \in V_N$.† In order to complete the definition of $\mathcal{K}$ we must define the operations θ_p, for every $p \in P$.

Consider a production $p = A \to a_0 B_1 a_1 \ldots B_r a_r$, $r \geq 0$. The operation θ_p is a function θ_p: $K_{B_1} \times \ldots \times K_{B_r} \to K_A$ mapping $\langle \sigma.B_k \in [D^{\overline{B}_k} \to D^{\underline{B}_k}] > 1 \leq k \leq r$ into $\sigma.A \in [D^{\overline{A}} \to D^{\underline{A}}]$. The attributes $x.A \in D^{\overline{A}}$, $y.A \in D^{\underline{A}}$, $x.B_k \in D^{\overline{B}_k}$, $y.B_k \in D^{\underline{B}_k}$, $1 \leq k \leq r$ must satisfy the equations

$$\begin{aligned} y.A &= \sigma.A(x.A) \\ y.B_k &= \sigma.B_k(x.B_k) \ , \ 1 \leq k \leq r. \end{aligned} \tag{3}$$

in addition to equations (1) and (2) of Section 3. The equations (1)-(3) uniquely determine the operation θ_p as follows.

Substituting (3) into (1) and (2) yields

$$x.B_k = F_{pk}(x.A, \sigma.B_1(x.B_1), \ldots, \sigma.B_r(x.B_r)), \ 1 \leq k \leq r \tag{4}$$

$$\sigma.A(x.A) = G_p(x.A, \sigma.B_1(x.B_1), \ldots, \sigma.B_r(x.B_r)). \tag{5}$$

Let $\langle E_1(x.A), \ldots, E_r(x.A) \rangle$ be the least fixed-point solution of system (4) with respect to $x.B_k$, $1 \leq k \leq r$ (see Section 4.4). Substituting it into (5) gives

$$\sigma.A(x.A) = G_p(x.A, \sigma.B_1(E_1(x.A)), \ldots, \sigma.B_r(E_r(x.A))),$$

thus defining θ_p as

$$\begin{aligned} \theta_p(\sigma.B_1, \ldots, \sigma.B_r) &= \sigma.A \\ &= \lambda x.A \in D^{\overline{A}} \cdot G_p(x.A, \sigma.B_1(E_1(x.A)), \ldots, \sigma.B_r(E_r(x.A))). \end{aligned} \tag{6a}$$

† Note that for A=S, $S = \{\varepsilon\}$ and thus $K_S = [\{\varepsilon\} \to D^{\underline{S}}] \cong D^{\underline{S}}$.

When r=0 (terminal production) we obtain the simpler definition

(6b) $$\theta_p = G_p.$$

The above construction may be thought of as a procedure for transforming a given K-system with both inherited and synthesized attribute into a new, purely synthesized K-system. This new K-system has (only synthesized) attributes $\sigma.A$, $\sigma.B_1,\ldots,\sigma.B_r$ and semantic rules defined by (6a) and (6b).

Definition 4.1 is now completed by noting that the construction of the G-algebra $\mathcal{K}$ and the initiality of T_G guarantee the existence of a unique homomorphism $h: T_G \to \mathcal{K}$, $h = \{h_A \mid A \in V_N\}$, which maps derivation trees on G into their semantics, where

$$h_A: T_{G,A} \to [D^{\overline{A}} \to D^{\underline{A}}], \qquad A \in V_N,\ A \neq S,$$

$$h_S: T_{G,S} \to D^{\underline{S}}.$$

The reader might now expect to see a proof that the algebraic definition of K-systems is equivalent to Knuth's original method of semantic definition, perhaps as a theorem of the form

> "For any $t \in T_{G,S}$, $h_S(t) = y.S$, where $y.S \in D^{\underline{S}}$ are the synthesized attribute values obtained using Knuth's method."

However, K-systems as presented in [14, 15, 16] give only an informal idea about how attribute values are to be computed. Consequently, the above theorem has no mathematical meaning. The purpose of the algebraic definition given in this section is to provide a formal treatment of this aspect of K-systems. Of course, one may use the necessary tree manipulation algorithms, as suggested by Knuth, and give an algorithm for attribute computation. This algorithm, which would have to be accepted on intuitive grounds, could then

be used as an "operational" definition of K-systems. In this case we can raise the question of proving a theorem relating the algebraic approach to the "operational" approach.† We leave this possibility open. Instead, we claim that the algebraic definition is the formalization of Knuth's method.

4.4 Fixed-point computation of attribute values

We now examine the fixed-point computation implied by the solution of equations (4). Equations (4) can be viewed as a system of equations in unknowns $x.B_k$, $1 \le k \le r$, with $x.A$ as a parameter. Intuitively, we are saying that given inherited attributes $x.A$ at node A we must be able to compute the inherited attributes $x.B_k$ at node B_k, $1 \le k \le r$. This is the place where the hypothesis about D being a continuous algebra plays an important role. For a fixed $x.A \in D^{\overline{A}}$, the right hand side of equations (4) gives the functions

$$F_{pk}^{x.A}: D^{\overline{B}_1} \times \ldots \times D^{\overline{B}_r} \to D^{\overline{B}_k}, \quad 1 \le k \le r,$$

defined as

$$(7) \qquad F_{pk}^{x.A}(x.B_1, \ldots, x.B_r) = F_{pk}(x.A, \sigma.B_1(x.B_1), \ldots, \sigma.B_r(x.B_r)).$$

The functions $F_{pk}^{x.A}$ are continuous and therefore the functions

$$(8) \qquad H_{x.A} = \langle F_{p1}^{x.A}, \ldots, F_{pr}^{x.A} \rangle$$

$$H_{x.A}: D^{\overline{B}_1} \times \ldots \times D^{\overline{B}_r} \to D^{\overline{B}_1} \times \ldots \times D^{\overline{B}_r}$$

are continuous for every $x.A \in D^{\overline{A}}$. Let $X = D^{\overline{B}_1} \times \ldots \times D^{\overline{B}_r}$. Then $H_{x.A}: X \to X$, as a continuous function, has a least fixed-point in X defined

† This is another instance of a well known problem: proving the equivalence of operational and mathematical semantics. An excellent analysis of this problem can be found in [10].

as

$$\text{Fix}_X(H_{X.A}) = \bigsqcup_{i=0}^{\infty} H^i_{X.A}(\perp). \tag{9}$$

Therefore the solutions $E: D^{\overline{A}} \to X$ of system (4) are $E = \lambda x \in D^{\overline{A}}.\text{Fix}_X(H_X)$. We know that with Knuth's approach, even if there is a circularity in the definition, the attribute computation must terminate (unsuccessfully) with some attributes being undefined. It is disturbing therefore to have introduced the least fixed-point function Fix_X in equation (8) since this function, from a computational point of view, may lead to a nonterminating computation.

We now briefly argue that the least fixed-point computation always terminates in a predetermined number of steps and, moreover, it suggests an algorithm for computing attributes in derivation trees. Given a continuous function $f: Y \to Y$ on any strict domain Y the least fixed-point of f, $\text{Fix}_Y(f)$, is the least upper bound of the ascending sequence of elements in Y

$$\perp = f^0(\perp) \leq f^1(\perp) \leq f^2(\perp) \leq \ldots.$$

This property suggests a computational procedure for $\text{Fix}_Y(f)$ (provided that everything else is computable in some sense): iterate the application of f to itself, starting from the element $\perp \in Y$. This computation may or may not terminate, and in this sense, Fix_Y is a partial function. However if Y is an elementary domain (has no infinite chains) then the computation always terminates (if $f^i(\perp) = f^{i+1}(\perp)$ for some i then $\text{Fix}_Y(f) = f^i(\perp)$).

The domain $X = D^{\overline{B}_1} \times \ldots \times D^{\overline{B}_r}$ is elementary since it is a cartesian product of elementary domains. If we evaluate the height of X we obtain an upper bound for the number of steps in which the computation of $\text{Fix}_X(H_{X.A})$ terminates. The sets D_s, $s \in \mathcal{S}$, have been made into strict elementary domains by adding the elements $\perp_{D_s}$, and by introducting the order relation $\leq$ defined

in Section 2.2. The height of each D_s is $ht(D_s) = 2$. If the length of $\overline{B}_k$ is ℓ_k, $1 \le k \le r$, then the height of $D^{\overline{B}_k}$ is

$$ht(D^{\overline{B}_k}) = \ell_k + 1 \quad \text{(the height of } \{\varepsilon\} \text{ is 1)}$$

and therefore the height of X is

$$ht(X) = \sum_{k=1}^{r} (\ell_k + 1) - r + 1 = \sum_{k=1}^{r} \ell_k + 1.$$

The longest chain in X cannot have more than ht(X) elements. Discounting $\perp$, which is an initialization step in the least fixed-point computation, it follows that the computation must terminate in $\ell = \sum_{k=1}^{r} \ell_k$ steps. We will see immediately that there is no coincidence that ℓ is also the total number of all inherited attributes of $B_1, \ldots, B_r$.

We now carry out a few steps in the computation of $Fix_X(H_{X.A})$ given by formula (9). We write $H^i_{X.A}(\perp)_k$ for the k-th projection of $H^i_{X.A}(\perp)$, $1 \le k \le r$.

The <u>first step</u> (i=1) of the least fixed-point computation gives

$$(10) \qquad H_{X.A}(\perp)_k = F_{pk}(x.A, \sigma.B_1(\perp), \ldots, \sigma.B_r(\perp)), \quad 1 \le k \le r.$$

Let $y_1.B_k = \sigma.B_k(\perp)$ and $x_1.B_k = H_{X.A}(\perp)_k$, $1 \le k \le r$. The values $y_1.B_k$ represent the values of the synthesized attribute of node B_k when all its inherited attributes are undefined (i.e., equal to $\perp$). $y_1.B_k$ is not necessarily equal to $\perp$, since some synthesized attributes may not depend on the inherited attributes (they may be constants). Formula (10) indicates that the values $x_1.B_k$ are those of the inherited attributes of node B_k which can be computed from the given x.A and $y_1.B_j$, $1 \le j \le r$. Of course, those attributes which are not (yet) evaluatable (i.e., they depend on undefined values) will be undefined.

The second step (i=2) of the iteration gives

$$(11) \qquad H^2_{x.A}(\perp)_k = H_{x.A}(H_{x.A}(\perp))_k$$
$$= F_{pk}(x.A, \sigma.B_1(x_1.B_1),\ldots, \sigma.B_r(x_1.B_r)), \ 1\leq k\leq r.$$

Let $y_2.B_k = \sigma.B_k(x_1.B_k)$ and $x_2.B_k = H^2_{x.A}(\perp)_k$, $1\leq k\leq r$. The values $y_2.B_k$ are new values of the synthesized attributes of B_k computed from the previously obtained inherited attribute values $x_1.B_k$. Since now some of $x_1.B_k$ may have defined values ($\neq \perp$), we may assume that more values in $y_2.B_k$ are defined. The values $y_2.B_k$ together with the values x.A are used in formula (11) to obtain new inherited attribute values $x_2.B_k$.

Each successive iteration (i = 3,...) computes new synthesized attribute values $y_3.B_k = \sigma.B_k(x_2.B_k),\ldots,$ and new inherited attribute values $x_3.B_k = H^3_{x.A}(\perp)_k,\ldots$. Each iteration uses the previous values to obtain new ones while the values x.A remain constant. Thus we obtain the following sequence of attribute values

$$(12) \qquad \perp \leq x_1.B_k \leq x_2.B_k \leq x_3.B_k \leq \ldots,$$

$$(13) \qquad \perp \leq y_1.B_k \leq y_2.B_k \leq y_3.B_k \leq \ldots,$$

for every $1\leq k\leq r$. The inequalities in (12) are the order relation on $D^{\overline{B}_k}$ and those in (13) are the order relation on $D^{\underline{B}_k}$. These inequalities result from the fact that the functions $\sigma.B_k$ and F_{pk} used in the computation of the above sequences are monotonic. This fact turns out to be very important.

We know that in Knuth's method, once an attribute is computed its value cannot be changed, and it is never "recomputed" as in our scheme. What we need here is the property that once an attribute receives a defined value

(i.e., = $\perp$) it must remain that way no matter how many times its value is recomputed. Only the value of undefined attributes may change from one iteration to another.

The inequalities (12) and (13) guarantee that our computation has this property. The inequality $y_1.B_k \leq y_2.B_k$ interpreted in the domain D^{B_k} means that the B_k-tuples $y_1.B_k$ and $y_2.B_k$ are such that they may differ from one another only in the undefined components of $y_1.B_k$. This also explains why the computation must terminate; once an attribute is defined it must remain that way in all subsequent steps, and there is only a finite number of attributes. Since the iteration is performed on inherited attributes, their number imposes an upper bound on the number of iterations. The same result was obtained previously from other considerations.

The fixed-point computation described previously can be very easily made into an algorithm for computing attributes on derivation trees. The algorithm terminates even if definitions are circular. In this case, some attributes will remain undefined. Unfortunately, the remarkable simplicity of an algorithm based on the fixed-point computation is overshadowed by its inefficiency, so presently, we see no practical applications for such an algorithm. However, there are some practical conclusions which can be drawn from this approach which are discussed in Section 5.

4.5 Example

Consider Knuth's binary number K-system [14, p. 130].

The original K-system:

Production	Equation (2)†	Equations (1)†
1. $B \to 0$	$v.B = 0$	
2. $B \to 1$	$v.B = 2 \uparrow s.B$	
3. $L \to B$	$v.L = v.B$	$s.B = s.L$
	$\ell.L = 1$	
4. $L_1 \to L_2 B$	$v.L_1 = v.L_2 + v.B$	$s.L_2 = s.L_1 + 1$
	$\ell.L_1 = \ell.L_2 + 1$	$s.B = s.L_1$
5. $N \to L$	$v.N = v.L$	$s.L = 0$
6. $N \to L_1 \cdot L_2$	$v.N = v.L_1 + v.L_2$	$s.L_1 = 0$
		$s.L_2 = -\ell.L_2$

Attribute s is inherited; v and ℓ are synthesized. The Σ-algebra D for this K-system (Definition 4.1) consists of sorts $\mathcal{S}$ = {int, ral}, carriers $D_{int} = Z$ (the integers) and $D_{ral} = Q$ (the rationals), and the operations of binary addition and unary subtraction on integers, binary addition on rationals, exponentiation of rationals by integers to yield rationals, and constants $0,1 \in Z \cap Q$. The attribute type strings for each nonterminal are

Inherited	Synthesized
$\overline{B}$ = int	B = ral
$\overline{L}$ = int	L = ral int
$\overline{N} = \varepsilon$	N = ral

The attribute value domains (and the variable names ranging over these domains) are

†as defined in Section 3.

$D^{\overline{B}} = Z$ (s.B)

$D^{\overline{L}} = Z$ ($s.L, s.L_1, s.L_2$)

$D^{\overline{N}} = \{\varepsilon\}$

$D^{\underline{B}} = Q$ (v.B)

$D^{\underline{L}} = Q \times Z$ ($\langle v.L, \ell.L\rangle, \langle v.L_1, \ell.L_1\rangle, \langle v.L_2, \ell.L_2\rangle$)

$D^{\underline{N}} = Q$ (v.N)

The semantic functions need not be made explicit since their domain is implied by Definition 4.1, and they are given by the semantic equations interpreted in the algebra D.

In order to obtain the new purely synthesized K-system and the associate G-algebra $\mathcal{K}$, we first assume that Z and Q have been replaced by $Z_\perp$ and $Q_\perp$, and that the operations of addition, etc. have been replaced by their strict extensions. The carriers $K_A = [D^{\overline{A}} \to D^{\underline{A}}]$, $A \in V_N$, of $\mathcal{K}$ are

$$K_N = [\{\varepsilon\} \to Q_\perp] \cong Q_\perp$$

$$K_L = [Z_\perp \to Q_\perp \times Z_\perp] \cong [Z_\perp \to Q_\perp] \times [Z_\perp \to Z_\perp]$$

$$K_B = [Z_\perp \to Q_\perp],$$

where $\cong$ denotes isomorphism of complete posets. We use $K_N = Q_\perp$ and $K_L = [Z_\perp \to Q_\perp] \times [Z_\perp \to Z_\perp]$ for convenience, to avoid the use of projection function. In its semantic equations, the new purely synthesized K-system uses the notation $\sigma_v.N \in K_N$; $\sigma_v.L, \sigma_v.L_1, \sigma_v.L_2 \in [Z_\perp \to Q_\perp]$; $\sigma_\ell.L, \sigma_\ell.L_1, \sigma_\ell.L_2 \in [Z_\perp \to Z_\perp]$; $\sigma_v.B \in K_B$.

The new purely synthesized K-system:

1. $B \to 0$ $\quad \sigma_v.B = \lambda s.0$

2. $B \to 1$ $\quad \sigma_v.B = \lambda s.2 \uparrow s$

3. $L \to B$ $\quad \sigma_v.L = \lambda s.(\sigma_v.B(s))$
$\quad \sigma_\ell.L = \lambda s.1$

4. $L_1 \to L_2 B$ $\quad \sigma_v.L_1 = \lambda s.(\sigma_v.L_2(s+1) + \sigma_v.B(s))$
$\quad \sigma_\ell.L_1 = \lambda s.(\sigma_v.L_2(s+1) + 1)$

5. $N \to L$ $\quad \sigma_v.N = \sigma_v.L(0)$

6. $N \to L_1 \cdot L_2$ $\quad \sigma_v.N = \sigma_v.L_1(0) + \sigma_v.L_2(\mathrm{Fix}_{Z_\perp}(-\sigma_\ell.L_2))$

We now show how to obtain the new semantic equation for production 6. Equations (4) corresponding to production 6 of the original K-system are

$$(9) \qquad \begin{aligned} s.L_1 &= 0 \\ s.L_2 &= -\sigma_\ell.L_2(s.L_2) \end{aligned}$$

the solution of which is

$$\langle E_1, E_2 \rangle = \langle 0, \mathrm{Fix}_{Z_\perp}(-\sigma_\ell.L_2) \rangle \in Z_\perp \times Z_\perp.$$

Thus $\sigma_v.N = \sigma_v.L_1(E_1) + \sigma_v.L_2(E_2)$, which yields the desired equation. The new semantic equations for the other productions are obtained similarly. The associated G-algebra operations are

$$\theta_1 = \lambda s.0$$

$$\theta_2 = \lambda s.2 \uparrow s$$

$$\theta_3(v) = \lambda s.\langle v(s), 1 \rangle$$

$$\theta_4(v_1, \ell_1, v_2) = \lambda s.\langle v_1(s+1) + v_2(s), \ell_1(s+1) + 1 \rangle$$

$$\theta_5(v, \ell) = v(0)$$

$$\theta_6(v_1, \ell_1, v_2, \ell_2) = v_1(0) + v_2(\mathrm{Fix}_{Z_\perp}(-\ell_2)),$$

where $v, v_1, v_2 \in [Z_\perp \to Q_\perp]$, $\ell, \ell_1, \ell_2 \in [Z_\perp \to Z_\perp]$, and $s \in Z_\perp$.

The meaning of the string 1101.01, whose derivation tree is

$$t = \pi_6\underline{(}\pi_4\underline{(}\pi_4\underline{(}\pi_4\underline{(}\pi_3\underline{(}\pi_2\underline{)}\pi_2\underline{)}\pi_1\underline{)}\pi_2\underline{)}\pi_4\underline{(}\pi_3\underline{(}\pi_1\underline{)}\pi_2\underline{))} \text{ , is}$$

$$h_N(t) = \theta_6(\theta_4(\theta_4(\theta_4(\theta_3(\theta_2),\theta_2),\theta_1),\theta_2),\theta_4(\theta_3(\theta_1),\theta_2)) \text{ ,}$$

which evaluates to $h_N(t) = 2^3 + 2^2 + 2^0 + 2^{-2} = 13.25$. Details are left to the reader.

5. Some practical considerations

The algebraic definition of K-systems has some important consequences concerning the methodology for developing semantic definitions in general, and the use of K-systems in particular.

First of all, since we are convinced that the algebraic formulation of K-systems is at least as powerful as Knuth's original approach, we can work directly in the algebraic framework as suggested in [12]. There is no loss in power or convenience, and by making evident the algebraic structure of the semantic domain, our understanding can only gain from such definitions. Moreover, the algebraic approach has none of the "well-definability" problems which appear when using Knuth's approach, and we also benefit from established proof techniques available in the algebraic method.

An even better approach is the combined use of both methods, algebraic and Knuth's, for the following reasons. The algebraic method, however precise and elegant, offers little aid to the intuition, since it is not always obvious which semantic algebra is required in a particular application. On the other hand, Knuth's approach is more intuitive, especially when computability is of great concern, such as in translator applications. In these cases Knuth's approach should help us discover what kind of algebra is needed in a particular application. It is a definite aid to the intuition if we can think in terms of attribute values being passed up and down in derivation trees and write our intuition in a precise algebraic manner.

There is an important observation to be made concerning the combined use of the Knuthian and algebraic approaches. We do not recommend first writing a Knuthian definition and then trying to express it algebraically as in Section 4.3. This was done only to show that any Knuthian definition can be given an algebraic formulation, including circular K-systems. This

led to recursive systems of equations which in actual practice can and should be avoided in the interest of simpler definitions.

The recursive nature of system (4) in Section 4.3 has two origins: one in the fact that Knuthian definitions can be circular, the other in our choice of carriers as $K_A = [D^{\overline{A}} \rightarrow D^{\underline{A}}]$, $A \in V_N$. Assuming carriers of this form may lead to recursive systems even though the definition is not circular. This happens because we are assuming that every synthesized attribute of A is a function of all inherited attributes of A. In practice, however, not all of this functional dependency may actually exist. To determine the actual dependency, the following definition is useful:

> Let $x_i.A$ be an inherited, and $y_j.A$ a synthesized attribute of $A \in V_N$. We say that $y_j.A$ depends on $x_i.A$ iff there exists a derivation tree with root A such that there exists a path from $x_i.A$ to $y_j.A$ in the attribute dependency graph on that tree.

Determining, for each $A \in V_N$, the dependency of the synthesized attributes of A upon the inherited attributes of A requires an algorithm very similar to Knuth's K-system circularity detection algorithm. Once the dependencies were determined, the algebraic carriers could be defined accordingly. Unfortunately, such an algorithm has exponential complexity [13] and therefore may not be practical.

We advocate a different approach which imposes a reasonable additional requirement on the K-system specifier. In addition to the requirements of Definition 4.1, the specification of a K-system would include for each $A \in V_N$ a specification of the dependency of the synthesized attributes of A upon the inherited attributes of A, expressed as a directed graph d_A. The nodes of d_A are labeled with the attributes of A, and an arc $\langle x_i.A, y_j.A \rangle$ is contained in d_A iff $y_j.A$ depends on $x_i.A$. This information is well

known to the system specifier, since he is well aware of the desired dependency between attributes in his K-system definition.

The dependency graphs d_A, $A \in V_N$, and Knuth's graphs D_p, $p \in P$ [14, p. 135] can be used to check the consistency and completeness of a K-system definition as follows.

For each production (say the p-th) $A \to a_0 B_1 \cdots B_r a_r$:

(a) Construct the directed graph union

$$d_p = D_p \cup d_{B_1} \cup \cdots \cup d_{B_r};$$

(b) Consistency Check: d_p is acyclic;

(c) Completeness Check: If there exists a path from $x_i.A$ to $y_j.A$ in d_p, then there must exist an arc $\langle x_i.A, y_j.A \rangle$ in d_A.

This algorithm has the same complexity (considerably less than exponential [8]) as a transitive closure algorithm.

Alternatively, in an algebraic definition of a K-system, the consistency check can be expressed as the condition that the equation systems (4) of Section 4.3 can be solved by successive elimination of the variables $x.B_k$, $1 \le k \le r$, without recourse to fixed-point operations. In this case, we do not need to extend the semantic domains D_s, $s \in \mathcal{A}$, to complete posets and the primitive semantic functions $f \in \Sigma_{w,s}$, $\langle w,s \rangle \in \mathcal{A}^* \times \mathcal{A}$, to continuous functions.

5.1 Example

We now repeat parts of Example 4.5 in light of the foregoing discussion. In addition to the previous specification, the dependency specification is

This specification is consistent with the semantic equations of the original K-system. In particular, it indicates that $\ell.L$ does not depend on $s.L$, and therefore, the K-system is nonrecursive, as we shall see.

The new purely synthesized K-system and associated G-algebra $\mathcal{K}'$ differ from what was previously obtained only in the parts affected by the dependency specification, namely the carriers of $\mathcal{K}'$, and the algebraic operations associated with productions 3, 4, and 6. According to the dependency specification, the carriers of $\mathcal{K}'$ are

$$K'_N = Q$$

$$K'_L = [Z \to Q] \times Z$$

$$K'_B = [Z \to Q].$$

We now rederive the semantic equation associated with production 6; productions 3 and 4 are treated similarly. Since $\ell.L$ does not depend on $s.L$, then neither does $\sigma_\ell.L$, i.e., $\sigma_\ell.L \in Z$, and thus the equations corresponding to (9) become

$$\text{(10)} \qquad \begin{aligned} s.L_1 &= 0 \\ s.L_2 &= -\sigma_\ell.L_2 \ . \end{aligned}$$

The solution of (10) is $\langle E_1, E_2 \rangle = \langle 0, -\sigma_\ell.L_2 \rangle$, obtained without recourse to fixed-point methods, and thus we obtain the following purely synthesized K-system.

1.	$B \to 0$	$\sigma_v.B = \lambda s.0$
2.	$B \to 1$	$\sigma_v.B = \lambda s.2\uparrow s$
3.	$L \to B$	$\sigma_v.L = \lambda s.(\sigma_v.B(s))$ $\sigma_\ell.L = 1$
4.	$L_1 \to L_2 B$	$\sigma_v.L_1 = \lambda s.(\sigma_v.L_2(s+1) + \sigma_v.B(s))$ $\sigma_\ell.L_1 = \sigma_\ell.L_2 + 1$
5.	$N \to L$	$\sigma_v.N = \sigma_v.L(0)$
6.	$N \to L_1 \cdot L_2$	$\sigma_v.N = \sigma_v.L_1(0) + \sigma_v.L_2(-\sigma_\ell.L_2)$

The operations of the algebra $\mathcal{K}'$ are

$$\theta_1' = \lambda s.0$$

$$\theta_2' = \lambda s.2\uparrow s$$

$$\theta_3'(v) = \langle v, 1\rangle$$

$$\theta_4'(v_1, \ell_1', v_2) = \langle \lambda s.(v_1(s+1) + v_2(s)), \ell_1' + 1\rangle$$

$$\theta_5'(v,\ell') = v(0)$$

$$\theta_6'(v_1,\ell_1', v_2, \ell_2') = v_1(0) + v_2(-\ell_2'),$$

where $v, v_1, v_2 \in [Z \to Q]$ and $\ell', \ell_1', \ell_2', s \in Z$.

6. Equivalence of K-Systems

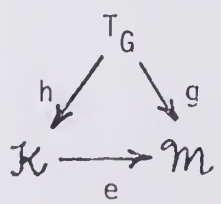
In this section we demonstrate the usefulness and elegance of the algebraic formulation of K-systems by applying it to the problem of proving two K-systems equivalent, in the sense of Definition 3.2. Our method applies only to K-systems that have the same underlying grammar. Carrying out such equivalence proofs may require proving that one of the given K-systems possesses certain properties. To this end we formulate an induction principle for many-sorted algebras. All of the foregoing concepts are illustrated by an example.

We first present our method for proving two K-systems with the same underlying grammar equivalent in the sense of Definition 3.2. The method is given and justified by the following theorem.

6.1 Theorem

Let K and M be K-systems with the same underlying grammar $G = \langle V_N, V_T, P, S \rangle$, and associated semantic algebras $\mathcal{K}$ and $\mathcal{M}$, respectively. We assume that $K_S = M_S$, i.e., the carriers of sort S in algebras $\mathcal{K}$ and $\mathcal{M}$ are identical. Then, K and M are equivalent (in the sense of Definition 3.2) if there exists a homomorphism $e: \mathcal{K} \to \mathcal{M}$ such that e_S is the identity function on K_S (i.e., $e_S = 1_{K_S}$).

Proof: Since T_G is initial, there exist unique G-homomorphisms $h: T_G \to \mathcal{K}$ and $g: T_G \to \mathcal{M}$. Since the composition of G-homomorphisms is also a G-homomorphism [12], we have $e \circ h = g$, i.e., the diagram



commutes. Thus for all $t \in T_{G,S}$, $e_S(h_S(t)) = g_S(t)$, which implies for all $t \in T_{G,S}$, $h_S(t) = g_S(t)$ (since $e_S = 1_{K_S}$), establishing the equivalence of K and M. ☒

We now illustrate the use of Theorem 6.1 via an example.

6.2 Example

Consider yet another K-system for the semantics of binary numbers, represented by

Productions	Semantic Rules
1. $B \to 0$	$v.B = 0$
	$\ell.B = 1$
2. $B \to 1$	$v.B = 1$
	$\ell.B = 1$
3. $L \to B$	$v.L = v.B$
	$\ell.L = \ell.B$
4. $L_1 \to L_2B$	$v.L_1 = 2 * v.L_2 + v.B$
	$\ell.L_1 = \ell.L_2 + 1$
5. $N \to L$	$v.N = v.L$
6. $N \to L_1.L_2$	$v.N = v.L_1 + v.L_2 * 2\uparrow(-\ell.L_2)$

This K-system is purely synthesized and has the same underlying grammar as those of Examples 4.5 and 5.1. Its Σ-algebra D consists of sorts {int, ral}, carriers $D_{int} = Z$ (the integers) and $D_{ral} = Q$ (the rationals), and operations 0, 1, -, +, *, and $\uparrow$ as in Examples 4.5 and 5.1. The attribute type strings for each nonterminal are $\overline{B} = \overline{L} = \overline{N} = \varepsilon$, $\underline{B} = \underline{L} = \underline{ral}\ \underline{int}$, and $N = \underline{ral}$. The synthesized attribute value domains (and the variable names ranging over these

domains) are

$$D^{\underline{B}} = Q \times Z \qquad (\langle v.B,\ \ell.B\rangle)$$

$$D^{\underline{L}} = Q \times Z \qquad (\langle v.L,\ \ell.L\rangle,\ \langle v.L_1,\ \ell.L_1\rangle,\ \langle v.L_2,\ \ell.L_2\rangle)$$

$$D^{\underline{N}} = Q \qquad (v.N)$$

Let $\mathcal{M}$ denote the semantic G-algebra associated with the above K-system; the carriers of $\mathcal{M}$ are $M_B = M_L = Q \times Z$ and $M_N = Q$. The operations of $\mathcal{M}$ are

$$\tau_1 = \langle 0,1\rangle$$

$$\tau_2 = \langle 1,1\rangle$$

$$\tau_3(v,\ell) = \langle v,\ell\rangle$$

$$\tau_4(v_1,\ell_1,v_2,\ell_2) = \langle 2*v_1 + v_2,\ \ell_1 + 1\rangle$$

$$\tau_5(v,\ell) = v$$

$$\tau_6(v_1,\ell_1,v_2,\ell_2) = v_1 + v_2*2\uparrow(-\ell_2)$$

where $v,v_1,v_2 \in Q$ and $\ell,\ell_1,\ell_2 \in Z$.

We now prove the equivalence of the above K-system and the K-system of Example 5.1, whose semantic algebra is $\mathcal{K}'$ with carriers K'_N, K'_L, K'_B and operations $\theta'_1,\ldots,\theta'_6$ as defined in Example 5.1. Following Theorem 6.1, it is sufficient to define a mapping $e: \mathcal{K}' \to \mathcal{M}$, where $e = \{e_N, e_L, e_B\}$ and $e_N = 1_Q$, and to show that e is a G-homomorphism. To this end, define $e_A: K'_A \to M_A$, $A = N,L,B$ as

$$e_N = 1_Q$$

$$e_L: [Z \to Q] \times Z \to Q \times Z \qquad \underline{\text{where}} \quad e_L(f,z) = \langle f(0),\ z\rangle$$

$$e_B: [Z \to Q] \to Q \times Z \qquad \underline{\text{where}} \quad e_B(f) = \langle f(0),\ 1\rangle.$$

To show that e is a G-homomorphism, the following analysis by cases must be performed.

(a1) SHOW $e_B(\theta_1') = \tau_1$.

LHS = $e_B(\lambda s.0) = \langle \lambda s.0(0), 1\rangle = \langle 0, 1\rangle$

RHS = $\langle 0, 1\rangle$.

(a2) SHOW $e_B(\theta_2') = \tau_2$

LHS = $e_B(\lambda s.2\uparrow s) = \langle \lambda s.2\uparrow s(0), 1\rangle = \langle 1, 1\rangle$

RHS = $\langle 1, 1\rangle$.

(a3) SHOW $e_L(\theta_3'(v)) = \tau_3(e_B(v))$

LHS = $e_L(v, 1) = \langle v(0), 1\rangle$

RHS = $\tau_3(v(0), 1) = \langle v(0), 1\rangle$.

(a4) SHOW $e_L(\theta_4'(v_1, \ell_1', v_2)) = \tau_4(e_L(v_1, \ell_1'), e_B(v_2))$

LHS = $e_L(\lambda s.(v_1(s+1) + v_2(s)), \ell_1' + 1)$

$= \langle \lambda s.(v_1(s+1) + v_2(s))(0), \ell_1' + 1\rangle$

$= \langle v_1(1) + v_2(0), \ell_1' + 1\rangle$

RHS = $\tau_4(v_1(0), \ell_1', v_2(0), 1)$

$= \langle 2*v_1(0) + v_2(0), \ell_1' + 1\rangle$.

(a5) SHOW $e_N(\theta_5'(v,\ell')) = \tau_5(e_L(v,\ell'))$

LHS = $e_N(v(0)) = v(0)$

RHS = $\tau_5(v(0), \ell') = v(0)$.

(a6) SHOW $e_N(\theta_6'(v_1, \ell_1', v_2, \ell_2')) = \tau_6(e_L(v_1, \ell_1'), e_L(v_2, \ell_2'))$

LHS = $e_N(v_1(0) + v_2(-\ell_2')) = v_1(0) + v_2(-\ell_2')$

RHS = $\tau_6(v_1(0), \ell_1', v_2(0), \ell_2')$

$= v_1(0) + v_2(0)*2\uparrow(-\ell_2')$.

Identities (a1), (a2), (a3), and (a5) are clearly satisfied. To show the satisfaction of (a4) and (a6) it is sufficient to prove that

(11) $$(\forall z \in Z)(v_i(z) = v_i(0) * 2\uparrow z),$$

where $v_i \in [Z \to Q]$, and i=1 for (a4) and i=2 for (a6).

6.3 Remarks

Unfortunately, equation (11) is not satisfied for all $v_1, v_2 \in [Z \to Q]$. However, we need be concerned in (11) only with those values of v_1 and v_2 that can occur within the definition of the K-system of Example 5.1. More particularly, the values of v_1 in (a4) and v_2 in (a6) can arise only as the result of mapping L-rooted parse trees $t \in T_{G,L}$ into their semantic meanings via the unique homomorphism $h'\colon T_G \to \mathcal{K}'$, namely via $h'_L\colon T_{G,L} \to K'_L$.† Viewing (11) as a (total) predicate on $[Z \to Q]$, we must prove that for all $t \in T_{G,L}$, the function $v = pr_1(h'_L(t))$ satisfies (11), where pr_1 extracts the first component of $h'_L(t)$ (recall that $K'_L = [Z \to Q] \times Z$).

We now state and apply an induction principle which will prove useful in demonstrating such results.

6.4 Theorem (Principle of Structural Induction on Many-Sorted Algebras)

Let Σ be an $\mathcal{S}$-sorted operator domain, let X be any Σ-algebra, and let $h\colon T_\Sigma \to X$ be the unique Σ-homomorphism from T_Σ to X. Let $P = \{P_s\}_{s \in \mathcal{S}}$ be an $\mathcal{S}$-indexed family of total predicates on X, i.e., P_s is a total predicate on X_s for all $s \in \mathcal{S}$.

If (0) $(\forall s \in \mathcal{S})(\forall \sigma \in \Sigma_{\varepsilon,s})(P_s(\sigma_X))$

and (1) $(\forall \langle w,s\rangle \in \mathcal{S}^+ \times \mathcal{S})(\forall \sigma \in \Sigma_{w,s})(\forall x \in X^w)(\bigwedge P^w(x) \supset P_s(\sigma_X(x)))$

then (2) $(\forall s \in \mathcal{S})(\forall t \in T_{\Sigma,s})(P_s(h_s(t)))$,

where if $w = s_1 \cdots s_n$ and $x = \langle x_1, \ldots, x_n\rangle \in X^w$, $n>0$, then

$$\bigwedge P^w(x) \triangleq P_{s_1}(x_1) \wedge \cdots \wedge P_{s_n}(x_n).$$

Proof: The theorem follows directly from: (a) the predicates P can be viewed as a family of sets $P_s \subseteq X_s$ for all $s \in \mathcal{S}$; (b) (0) and (1), if true, express that P can be regarded as a subalgebra of X; (c) $h(T_\Sigma)$ is the least subalgebra

† We are interested in e being a homomorphism from $h'(T_G)$ into $\mathcal{M}$ rather than one from $\mathcal{K}'$ to $\mathcal{M}$.

of X (an easy consequence of the initiality of T_Σ); therefore for all $s \in \mathcal{S}$, $h(T_\Sigma)_s \subseteq P_s$, the desired conclusion. ☒

In particular, if for all $s \in \mathcal{S}$, P_s is a total predicate on $T_{\Sigma,s}$ satisfying (0) and (1), then $P_s(t)$ is true for all $s \in \mathcal{S}$ and $t \in T_{\Sigma,s}$, since $h: T_\Sigma \to T_\Sigma$ is the identity homomorphism.† We shall now apply Theorem 6.4 to complete the proof of Example 6.2.

6.5 Completion of Proof of Example 6.2

Define the following predicates on the G-algebra $\mathcal{K}'$ of Example 5.1.

(12a) $P_B(v) \triangleq (\forall z \in Z)(v(z) = v(0) * 2\uparrow z)$

(12b) $P_L(v,\ell) \triangleq (\forall z \in Z)(v(z) = v(0) * 2\uparrow z)$

(12c) $P_N(q) \triangleq$ true, where $v \in [Z \to Q]$, $\ell \in Z$, $q \in Q$.

Theorem 6.4 requires consideration of the following cases.

(b1) $P_B(\theta_1') = (\forall z \in Z)(\lambda s.0(z) = \lambda s.0(0) * 2\uparrow z)$

$= (\forall z \in Z)(0 = 0 * 2\uparrow z) =$ true

(b2) $P_B(\theta_2') = (\forall z \in Z)(\lambda s.2\uparrow s(z) = \lambda s.2\uparrow s(0) * 2\uparrow z)$

$= (\forall z \in Z)(2\uparrow z = (2\uparrow 0) * (2\uparrow z)) =$ true

(b3) Assume $P_B(v)$ for $v \in [Z \to Q]$. Then

$P_L(\theta_3'(v)) = P_L(v,1) = (\forall z \in Z)(v(z) = v(0) * 2\uparrow z) =$ true,

which follows directly from the hypothesis.

(b4) Assume $P_L(v_1,\ell_1')$ for $\langle v_1,\ell_1'\rangle \in [Z \to Q] \times Z$ and $P_B(v_2)$ for $v_2 \in [Z \to Q]$. Then

$P_L(\theta_4'(v_1,\ell_1',v_2)) = P_L(\lambda s.(v_1(s+1) + v_2(s)),\ \ell_1' + 1)$

$= (\forall z \in Z)(v_1(z+1) + v_2(z) = (v_1(1) + v_2(0)) * 2\uparrow z) =$ true,

which follows readily from the hypotheses.

Cases (b1) through (b4) and Theorem 6.4 thus establish that for all

† The usual mathematical induction can be obtained from the above principle by taking $\mathcal{S} = \{\underline{int}\}$, $\Sigma_{\epsilon,s} = \{0\}$, $\Sigma_{\underline{int},\underline{int}} = \{succ\}$ and $\Sigma_{w,s} = \Phi$ for all other $\langle w,s\rangle$.

$t \in T_{G,L}$, $P_L(h'_L(t))$, which shows that $pr_1(h'_L(t))$ satisfies (11) of Section 6.3. Thus the K-systems of Examples 5.1 and 6.2 are equivalent. ☒

In order to prove the equivalence of the K-systems of Examples 4.5 and 6.2, the algebraic carriers Z and Q of Example 6.2 must be extended to domains $Z_\perp$ and $Q_\perp$, and the primitive semantic functions must be replaced by their strict extensions, so that the two K-systems are comparable. Then the mappings $e = \{e_N, e_L, e_B\}$ such that

$$e_N = 1_{Q_\perp}$$

$$e_L: [Z_\perp \to Q_\perp] \times [Z_\perp \to Z_\perp] \to Q_\perp \times Z_\perp \quad \underline{\text{where}} \quad e_L(f,j) = \langle f(0), j(0)\rangle$$

$$e_B: [Z_\perp \to Q_\perp] \to Q_\perp \times Z_\perp \quad \underline{\text{where}}\ e_B(f) = \langle f(0), 1\rangle$$

can be defined. To show that e is a G-homomorphism reduces to proving that

$$\begin{aligned} &v_1(1) = 2 * v_1(0) \\ &\ell_1(1) = \ell_1(0) \\ &v_2(\mathrm{Fix}_{Z_\perp}(-\ell_2)) = v_2(0) * 2\uparrow(-\ell_2(0)) \end{aligned} \tag{13}$$

where the values of $\langle v_1, \ell_1\rangle$, $\langle v_2, \ell_2\rangle \in [Z_\perp \to Q_\perp] \times [Z_\perp \to Z_\perp]$ can arise only as the result of mapping parse trees $t \in T_{G,L}$ into their semantic meanings via $h_L: T_{G,L} \to K_L$. Properties (13) can be verified via Theorem 6.4 with predicates P_B on $[Z_\perp \to Q_\perp]$ and P_L on $[Z_\perp \to Q_\perp] \times [Z_\perp \to Z_\perp]$, where

$$P_B(v) = (\forall z \in Z)(v(z) = v(0) * 2\uparrow z)$$

$$P_L(v,\ell) = P_B(v) \wedge (\forall z \in Z)(\forall y \in Z)(\ell(z) = \ell(y)).$$

Details are left to the reader.

7. Conclusion

Knuthian semantic systems, or K-systems, are a useful and intuitively appealing method for specifying syntax-directed semantics of context-free languages. We have formalized, within the framework of initial algebra semantics, K-systems containing both inherited and synthesized attributes, thus combining the intuitive appeal of K-systems with the power and elegance of algebraic methods. Since Knuth's initial presentation of K-systems was informal, we have left open the problem of proving it equivalent to our algebraic formulation.†

Our formulation of K-systems enables proof of the properties of the semantics of context-free languages (such as correctness of translators and their implementation) to be conveniently carried out by induction on the (context-free) syntactic structure of the underlying grammar [4].

An important byproduct of our formulation is a method for transforming a K-system containing both inherited and synthesized attributes into an equivalent purely synthesized K-system. The new purely synthesized K-system has the same underlying grammar as the original, but its semantic domains are functions from the original inherited attribute semantic domains to synthesized attribute semantic domains.

This transformation generally requires the solution of recursive systems of equations. Such a solution is achieved via fixed-point methods with all their attendant mathematical apparatus, thus requiring the algebraic specification of the given K-system to be or extended to be continuous. For practical purposes we advocate the use of only those K-systems in which these systems of equations are nonrecursive, i.e., equations that can be solved by successive elimination of variables without recourse to fixed-point methods. We feel that such nonrecursive K-systems are sufficient for the

† In effect, we offered a "mathematical" definition in the sense that the algebraic semantics is independent of any algorithm for attribute computation. In contrast,

vast majority of practical applications.

To foster the use of these more practical K-systems, we have suggested that the specification of a K-system include for each $A \in V_N$ a specification of the dependency of the synthesized attributes of A upon the inherited attributes of A. The K-system can then be checked for "consistency" (the K-system is nonrecursive) and "completeness" (the K-system actually contains all the specified attribute dependencies) with respect to this specification.

We advocate the use of both Knuth's original approach and our algebraic formulation as a joint means of designing and defining K-systems for particular applications. Our intuition is better aided by the visualization of inherited and synthesized attribute values being computed and communicated over attribute dependency graphs superimposed on derivation trees (an advantage originally cited by Knuth [14]), whereas the algebraic formulation renders our intuition precise and enables us to use associated powerful proof techniques.

The usefulness of the algebraic formulation of K-systems was demonstrated via its application to proving the equivalence of K-systems having the same underlying grammar. It was found that such proofs may require verifying that a K-system possesses certain properties. Toward this latter end, a principle of structural induction on many-sorted algebras was formulated, justified, and applied.

---

† Knuth's method (as well as those in [3], [7], [18]) is "operational," i.e., the definition is given by a particular algorithm. A possible solution to proving the two definitions equivalent is to explicitely write Knuth's definition as an algorithm (program) and prove it consistent with our mathematical specification using, for instance, Floyd-Hoare method.

REFERENCES

[1] Berry, D.M. "On the Design and Specification of the Programming Language OREGANO," Computer Science Department, University of California, Los Angeles, UCLA-ENG-7388, January 1974.

[2] Birkhoff, G. and J.D. Lipson. "Heterogeneous Algebras," J. Combinatorial Theory, 8: 115-113, 1970.

[3] Bochmann, G.V. "Semantic Evaluation from Left to Right," Communications of the ACM, 19(2): 55-62, February 1976.

[4] Chirica, L.M. "Contributions to Compiler Correctness," Ph.D. dissertation, UCLA Computer Science Department, 1976.

[5] Culic, K. "Attributed Grammars and Languages," Department d' Informatique, Universite de Montreal, Montreal, 1969.

[6] Dreisbach, T.A. "A Declarative Semantic Definition of PL360," Computer Languages Group, Computer Science Department, University of California, Los Angeles, UCLA-ENG-7289, 1972.

[7] Fang, I. "FOLDS - A Declarative Formal Language Definition System," Computer Science Department, Stanford University, Palo Alto, California, STAN-CS-329, December 1972.

[8] Fischer, M.J. and A.R. Meyer. "Boolean Matrix Multiplication and Transitive Closure," Conference Record of the 12th IEEE Symposium on Switching and Automata Theory, East Lansing, Michigan, October 1971, IEEE, New York, 1971, pp. 129-131.

[9] Gerhart, S. "Correctness Preserving Program Transformations," Proceedings of the Second ACM Symposium on Principles of Programming Languages, Palo Alto, California, January 1975, ACM, New York, 1975, pp. 54-56.

[10] Gordon, M. "Models of Pure LISP," Ph.D. dissertation, Computer Science, Edinburgh University, Edinburgh, Scotland, 1973.

[11] Goguen, J.A. and J.W. Thatcher. "Initial Algebra Semantics," Proceedings of the 15th IEEE Symposium on Switching and Automata Theory, New Orleans, Louisiana, October 1974, IEEE, New York, 1974, pp. 63-77.

[12] Goguen, J.A., J.W. Thatcher, E.G. Wagner, and J.B. Wright. "Initial Algebra Semantics and Continuous Algebras," Journal of the ACM, 24(1): 68-95, January 1977.

[13] Jazayeri, M., W.F. Ogden, W.C. Rounds. "The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars," Communications of the ACM, 18(12): 697-706, December 1975.

[14] Knuth, D.E. "Semantics of Context-Free Languages," Math. Syst. Th. 2: 127-145, 1968.

[15] Knuth, D.E., "Semantics of Context-Free Languages: Correction," Math. Syst. Th. 5: 95-96, 1971.

[16] Knuth, D.E. "Examples of Formal Semantics," Lecture Notes in Math. No. 188, Springer-Verlag, Berlin, 1971, pp. 212-235.

[17] Koster, C.H.A. "Affix Grammars," ALGOL 68 Implementation, North-Holland, Amsterdam, 1971.

[18] Kennedy, K. and S.K. Warren. "Automatic Generation of Efficient Evaluators for Attribute Grammars," Conference Record of the Third ACM Symposium on Principles of Programming Languages, Atlanta, Georgia, January 1976, ACM, New York, 1976, pp. 32-49.

[19] Lewis, P.M. and B.K. Rosen. "Recursively Defined Data Types," Proceedings of ACM Symposium on Principles of Programming Languages, Boston, Massachusetts, October 1973, ACM, New York, 1973, pp. 125-138.

[20] Lewis, P.M., D.J. Rosenkrantz, and R.E. Stearns. "Attributed Translations," Journal of Computer and System Science, 9: 279-307, 1974.

[21] Lewis, P.M., D.J. Rosenkrantz, and R.E. Stearns. Compiler Design Theory, Addison-Wesley, Reading, Massachusetts, 1976.

[22] Markowsky, G. "Chain-Complete Posets and Directed Sets with Applications," IBM Thomas J. Watson Research Center, Yorkwown Heights, New York, Research Report RC5024, August 1974.

[23] Martin, D.F. and S.A. Vere. "On Syntax-Directed Transduction and Tree Transducers," Proceedings of the Second Annual ACM Symposium on Theory of Computing, Northampton, Massachusetts, May 1970, ACM, New York, 1970, pp. 129-135.

[24] Neel, D. and M. Amirchahy. "Semantic Attributes and Improvement of Generated Code," Proceedings of ACM National Conference, San Diego, California, November 1974, ACM, New York, 1974, pp. 1-10.

[25] Petrick, S.R. "Semantic Interpretation in the REQUEST System," IBM Thomas J. Watson Research Center, Yorktown Heights, New York, Research Report RC4457, July 1973.

[26] Scott, D. "Outline of a Mathematical Theory of Computation," Proceedings of the Fourth Annual Princeton Conference on Information Science and Systems, Princeton, New Jersey, 1970, Department of Electrical Engineering, Princeton University, Princeton, New Jersey, 1970, pp. 169-176.

[27] Scott, D. "Continuous Lattices," Computing Laboratory, Oxford University, Oxford, Great Britain, Technical Monograph PRG-7, 1971.

[28] Scott, D. "The Lattice of Flow Diagrams," Symposium on Semantics of Algorithmic Languages, edited by E. Engler, Lecture Notes in Mathematics No. 188, Springer-Verlag, New York, 1971, pp. 311-366.

[29] Scott, D. "Mathematical Concepts in Programming Language Semantics," Proceedings of the AFIPS Spring Joint Computer Conference, Atlantic City, New Jersey, May 1972, AFIPS Press, Montvale, New Jersey, 1972, pp. 225-234.

[30] Stearns, R.E. and P.M. Lewis. "Property Grammars and Table Machines," Information and Control, 14: 524-549, 1969.

[31] Wilner, W.T. "Declarative Semantic Definition," Computer Science Department, Stanford University, Palo Alto, California, STAN-CS-233-71, 1971.

[32] Wilner, W.T. "Formal Semantic Definition Using Synthesized and Inherited Attributes," Proceedings of Courant Institute Computer Science Symposium, New York, September 1970, Prentice-Hall, Englewood Cliffs, New Jersey, 1972, Vol. 2, pp. 25-39.

APPENDIX

How to Transform K-Systems to Satisfy Conditions (iii) and (iv) of Section 3

In this appendix we present a construction which transforms a K-system which does not satisfy conditions (iii) and (iv) of Section 3 into an equivaler one that does.

Let x and y be, respectively, the inherited and synthesized attributes of the given K-system, and let w and z be, respectively, new (distinct from x and y) inherited and synthesized attributes with the same value domains as x and y. Let $G = \langle V_N, V_T, P, S \rangle$ be the CFG underlying the given K-system and consider a typical (say the p-th) production

$$A \to a_0 B_1 a_1 \cdots B_r a_r ,$$

and the following four cases.

Case 1 $(A \neq S,\ r > 0)$

The given semantic equations in this case are

(A1.1) $\quad x.B_k = F_{pk}(x.A, y.B_1, \ldots, y.B_r, y.A, x.B_1, \ldots, x.B_r)$, $1 \leq k \leq r$

(A2.1) $\quad y.A = G_p(x.A, y.B_1, \ldots, y.B_r, y.A, x.B_1, \ldots, x.B_r)$.

Equations (A1) and (A2) do not satisfy conditions (iii) and (iv) of Section 3. Replace these semantic equations by

(A3a.1) $x.B_k = F_{pk}(x.A, y.B_1, \ldots, y.B_r, w.A, z.B_1, \ldots, z.B_r)$, $1 \le k \le r$

(A3b.1) $w.B_k = y.B_k$, $1 \le k \le r$

(A4a.1) $y.A = G_p(x.A, y.B_1, \ldots, y.B_r, w.A, z.B_1, \ldots, z.B_r)$

(A4b.1) $z.A = x.A$

Case 2 ($A = S$, $r > 0$)

The given semantic equations are

(A1.2) $x.B_k = F_{pk}(y.B_1, \ldots, y.B_r, y.S, x.B_1, \ldots, x.B_r)$, $1 \le k \le r$

(A2.2) $y.S = G_p(y.B_1, \ldots, y.B_r, y.S, x.B_1, \ldots, x.B_r)$.

The replacement for these semantic equations and corresponding production $S \to a_0 B_1 \cdots B_r a_r$ is not as simple as in Case 1. First, replace (A1) and (A2) with

(A3a.2) $x.B_k = F_{pk}(y.B_1, \ldots, y.B_r, w.S, z.B_1, \ldots, z.B_r)$, $1 \le k \le r$

(A3b.2) $w.B_k = y.B_k$, $1 \le k \le r$

(A4b.2) $y.S = G_p(y.B_1, \ldots, y.B_r, w.S, z.B_1, \ldots, z.B_r)$.

Note that there is no counterpart to equation (A4a.1). Let $T \notin V_N \cup V_T$ be a new distinguished symbol. Add the following production and semantic rules to the new K-system:

(A5.2) $T \to S$ $\quad w.S = y.S$

$\quad y.T = y.S$

Note that K-system fragment (A5.2) need be added only once to deal with all S-productions (Cases 2 and 4). Also note that the former distinguished symbol S now has inherited attributes w.

Case 3 ($A \neq S$, $r = 0$)

The semantic eqaation associated with $A \to a_0$, $A \neq S$, is

(A2.3) $$y.A = G_p(x.A, y.A).$$

Let $A' \notin V_N \cup V_T \cup \{T\}$ be a new nonterminal distinct from any other new nonterminal introduced in the process of transforming the original K-system into one that satisfies conditions (iii) and (iv) of Section 3. Replace $A \to a_0$ and (A2.3) by

(A6.3) $A \to A'$

$$x.A' = x.A$$
$$w.A' = y.A'$$
$$y.A = y.A'$$

(A7.3) $A' \to a_0$

$$y.A' = G_p(x.A', w.A').$$

Case 4 ($A = S$, $r = 0$)

This case is similar to Case 3, but with no attributes x.S and x.S'. The replacement K-system fragments are

(A6.4) $S \to S'$

$$w.S' = y.S'$$
$$y.S = y.S'$$

(A7.4) $S' \to a_0$

$$y.S' = G_p(w.S').$$

Of course fragment (A5.2) will "connect" (A6.4) and (A7.4) to the new distinguished symbol T.

Noting that w and x are inherited and y and z are synthesized attributes, it is easily seen that all versions of equations (A3) through (A7) in Cases 1 - 4 satisfy conditions (iii) and (iv) of Section 3. By constructing local attribute dependency graphs of the original K-system and corresponding graphs of the new K-system, it can be readily established that the two systems are equivalent.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-77-881	2.	3. Recipient's Accession No.
4. Title and Subtitle An Algebraic Definition of Knuthian Semantics			5. Report Date June 1977 6.
7. Author(s) Laurian M. Chirica and David F. Martin			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF Grant MCS 03633 ERDA Contract E(04-3)-34,
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC Energy Research and Development Administration Washington, DC			13. Type of Report & Period Covered PA 214 14.

15. Supplementary Notes

16. Abstracts

This paper presents an algebraic definition of Knuthian semantic systems (K-systems or attribute grammars) with both synthesized and inherited attributes. The approach is based on the initial algebra semantics principle formulated by Goguen, Thatcher, Wagner and Wright. Given a K-system semantic definition for a context-free grammar G, it is shown how to construct a many-sorted algebra $\mathcal{K}$ such that the semantic mapping from G-derivation trees into their "meaning" is the unique homomorphism from the initial algebra T_G of derivation trees into $\mathcal{K}$. The practical implications of the algebraic definition of K-systems are discussed, and the combined use of Knuth's original formulation and the algebraic approach for the development of semantic definitions is advocated. The usefulness of the algebraic formulation of K-systems is demonstrated by its application to proving the equivalence of K-systems having the same underlying grammar. It is shown that such proofs may require verifying that a K-system possesses certain properties. To this end, a principle of structural induction on many-sorted algebras is formulated, justified, and applied.

17. Key Words and Document Analysis. 17a. Descriptors

ttribute grammar
omplete poset
nitial algebra semantics
east fixed-points
any-sorted algebras
tructural induction

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price

FORM NTIS-35 (10-70)

USCOMM-DC 40329-P71